Polytechnic UNIVERSITY

Toward 10-100 Gbps Cryptographic Architectures

Ramesh Karri, Piyush Mishra, Igor Minkin
Kaiji Wu, Khary Alexander, Xuan Li

October 2002

WICAT TR 02-005

WICAT
WIRELESS INTERNET
Center for Advanced Technology

Towards 10-100 Gbps Cryptographic Architectures[1]

Khary Alexander[!], Ramesh Karri, Igor Minkin, Kaijie Wu, Piyush Mishra, Xuan Li
[!]IBM Corporation, Poughkeepsie, NY, 12601, kalexa01@utopia.poly.edu
ECE Department, Polytechnic University, 5 Metrotech Center, Brooklyn, NY, 11201
ramesh@india.poly.edu, iminki01, kwu03, pmishr01, xli03 @utopia.poly.edu

Abstract

Support for secure transactions over insecure public communication networks, built atop high-speed optical infrastructure, requires cryptographic primitives with 10-100Gbps throughput. In this paper we present various hardware architectures for high-speed block and stream ciphers and study the associated throughput and area trade-offs. We demonstrate high-speed encryption architectures for Advanced Encryption Standard (AES) based block and stream ciphers and SNOW stream cipher with 4.7Gbp, 4.6Gbps and 2.12Gbps throughput rates respectively. We then present our ongoing work on high-speed architectures for tree-based stream cipher Leviathan.

Keywords: cryptography, encryption, block-cipher, stream-cipher, FPGA, VHDL, AES, Rijndael, SNOW, Leviathan

1 Introduction

Demand for high-speed encryption is rapidly increasing due to the large bandwidth requirements of the evolving network applications. These applications include Virtual Private Network (VPN) aggregation points, secure e-commerce web servers and Local Area Networks (LANs) with aggregate security protocol offload. As a result, cryptographic processing in software is proving to be inadequate, thereby fueling the industry's push towards hardware implementations of various cryptographic architectures. In this paper we present various hardware architectures for high-speed block and stream ciphers and study the associated throughput vs. hardware area trade-offs.

Cryptographic algorithms are classified as hash, private-key and public key algorithms. Hash algorithms operate on arbitrary length messages to create a fixed length digest or hash. Public key algorithms, also known as asymmetric-key algorithms, use the basic idea of one-way functions to generate encryption-decryption key pair to be used for data encryption. On the other hand, functionality and security of private-key algorithms, also known as symmetric-key algorithms, depend on the single private key. Private-key algorithms are basically of two types, block cipher and stream cipher, and can operate in various modes of operation such as Electronic Book Code (ECB), Output Feed Back (OFB), and Cipher Block Chain (CBC) mode.

Block ciphers process plaintext (ciphertext) messages in discrete blocks and encryption (decryption) of a particular plaintext (ciphertext) with a block cipher results in the same ciphertext (plaintext) when the same key is used. Examples of block ciphers include Data Encryption Standard (DES), triple-DES (3DES), and AES finalist algorithms - Rijndael, Twofish, RC6 and Serpent. After a thorough study of security and performance benchmarks NIST chose Rijndael as the winner. Stream ciphers, on the other hand, are practical approximations to the *one time pad*, operating with a time-varying transformation on individual elements as small as single bits. With a stream cipher, the transformation of these smaller units will vary, depending on when they are encountered during the encryption process. Stream ciphers can be created from dedicated key stream generators, such as Linear Feedback Shift Register (LFSR) and tree-based stream ciphers or by certain modes of operation of block ciphers. Examples of stream ciphers are SNOW, Leviathan and Rijndael in Integer Counter Mode (ICM).

Several hardware implementations of cryptographic algorithms have been described in literature, such as the algorithm-specific implementations of DES [[8], [9], [10], [11]], IDEA [12], Twofish [12], and Blowfish [12] and hardware-software co-design based crypto-processors, such as CryptoManiac [13]. In this paper we present our work on various hardware architectures for high-speed block and stream ciphers that achieve 10 to 100 Gbps throughput rates and study the associated throughput and area trade-offs.

Section 2 describes the implementation results of the four AES [7] finalist block ciphers on Field Programmable Gate Arrays (FPGAs). It also presents various high-speed architectures for these symmetric block ciphers and validates these using AES (Rijndael). Section 3 discusses high-speed architectures for stream ciphers - AES in ICM, SNOW, and Leviathan - followed by the corresponding implementation results. Section 4 concludes this study.

[1] This work is supported by CISCO University Research Program

2 Block Ciphers

A symmetric block cipher encrypts plaintext by iteratively applying a round transformation to the input block using different round-keys for each round. Round keys are derived from the private key by using a key-schedule transformation. Decryption is the reverse process of encryption. Usually the complexity of encryption and decryption operations is more significant than that of key generation operation. Hence, in this paper we only focus on optimizing the operations used by data encryption and decryption for better system performance.

Figure 1 shows the architecture for symmetric Rijndael [1] encryption and decryption. During encryption input block is exclusive-or (XOR) with a round-key in the pre-processing step and the output is iteratively transformed N_r times using a round function where N_r denotes the total number of iterations and each iteration uses a different round key.

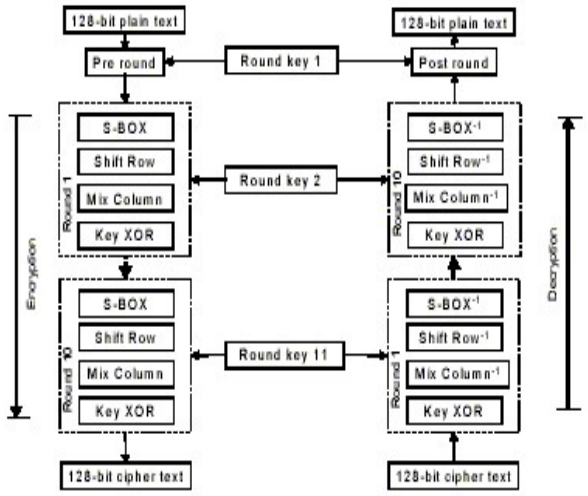

Figure 1: AES iterative block cipher

A round function consists of a series of operations (AES uses Substitution Box (SBox) - Shift Row (SRow) - Mix Column (MixCol) - Key XOR (KXor)). Number of rounds of encryption and decryption is a function of the data block size, user key size and the desired level of security. In most cases, increasing the number of encryption rounds improves security at the cost of system throughput and vice-versa. For example, AES processes data blocks of 128 bits, using cipher keys of 128, 192 or 256 bits. Table 1 shows the number of encryption/decryption rounds corresponding to these three different key sizes.

Table 1: Number of rounds of AES as a function of data blocks size and user key sizes

N_k	4	6	8
N_r	10	12	14
$N_b = 4$, N_k = key length/32, N_r = MAX (N_b, N_k) + 6			

2.1 FPGA Implementation of Block Ciphers

Table 2 shows the results of our implementation of four AES finalist block ciphers on Xilinx Virtex series XCV1000 FPGAs. All FPGA implementation were carried out using Synplicity® Synplify™ VHDL compiler, Modeltech® Modelsim™ VHDL simulator and Xilinx® Place and Route (PAR) tools. Details of these implementation results can be found in [3]. One Virtex slice contains two look-up tables (LUTs) and each LUT can implement four-input, one-output logic function. Throughput of a cipher represents the total number of bits encrypted per second and is calculated as *number of bits encrypted / (operation cycles * clock duration)*.

Table 2: Implementation results for AES finalists

	Area (Slices)	Frequency (MHz)	Throughput (Mbps)
Rijndael	3973	46.93	136.53
RC6	2397	23.99	73.11
Twofish	3262	20.16	75.9
Serpent	8073	28.638	57.28

An iterative looping architecture minimizes the hardware area requirements by implementing only one round over which the cipher iterates N_r times per block. This scheme however results in low throughput and can be costly in terms of the input and round key storage and multiplexing requirements. Throughput rates of all these implementations were less than 150 Mbps. Therefore advanced architectures need to be designed to achieve the target throughputs of 10Gbps and higher.

2.2 Advanced Architectures for Block Ciphers

Iterative block ciphers offer a variety of architectural options, each with the associated area and throughput tradeoffs. For example, a **loop-unrolled** architecture allows implementation of up to N_r rounds as a single combinational logic block, reducing the hardware for round key multiplexing and the number of clock cycles per block. However, this approach maximizes the hardware requirements and yields worst case register-register delay.

Another option for high-speed architectures is pipelining. A **partially pipelined** architecture achieves this by increasing the number of blocks of data that are being simultaneously operated upon. Each round is implemented as the atomic round element and the intermediate data are registered, decreasing the worst-case register-to-register delay. In the case of a full-length pipeline, the system will output an N-bit block at each clock cycle once the latency of the pipeline has been met. This architecture, however, also significantly increases the required hardware resources.

Basic pipelining can be extended by sub-pipelining to overcome these shortcomings. For example, AES can be sub-pipelined by partitioning the SRow and MixCol operations. This decreases the worst-case register-register

delay and increases the number of data blocks that can be operated on simultaneously by a factor equal to the number of sub-divisions. However, these benefits come at the cost of an increase in the number of clock cycles, thereby requiring a corresponding decrease in the worst-case delay between the stages.

2.3 FPGA Implementation of Advanced Architectures for AES Block Ciphers

After analyzing these various architectures we decided to optimize the sub-pipelined AES block cipher with 128-bit block and 128-bit user key size to investigate the associated throughput vs. area tradeoffs. An important observation regarding AES is that the most dominant operation of AES is the SBox since 128-bit data path requires sixteen copies of the 8-bit SBox [3]. As a result, round function was sub-pipelined to isolate it from SRow, MixCol and KXOR. Even though the second stage contained a multiplexer (to bypass MixCol during the last round of encryption) SBox operation still formed the critical path.

Two architectures were developed using this sub-pipelined round and in both the cases key scheduling was performed a priori. First architecture (SP_1_1) [2] is built around one round with one sub-pipeline register to split it into 2-stages. Figure 2 shows that round-keys were stored in a key-RAM and 128-bit *TextIn* port was used to supply both user key and plaintext. Second architecture is a five stage partially pipelined (SP_5_1) loop unrolling of the sub-pipelined round, as shown in Figure 3. First four rounds were optimized by removing the bypass multiplexer in the second stage of the sub-pipeline since MixCol is always performed in these stages. In this case round keys were stored in a file of registers. This architecture resulted in a 10 stage pipeline (5 rounds * 2 stages per round) with 2.1 cycles ($2 + {}^{1}/_{10}$) per block.

2.4 Comparison with previous work

Table 3 shows the implementation results for these two AES architectures on Xilinx XC2V2000BG575-5 FPGA.

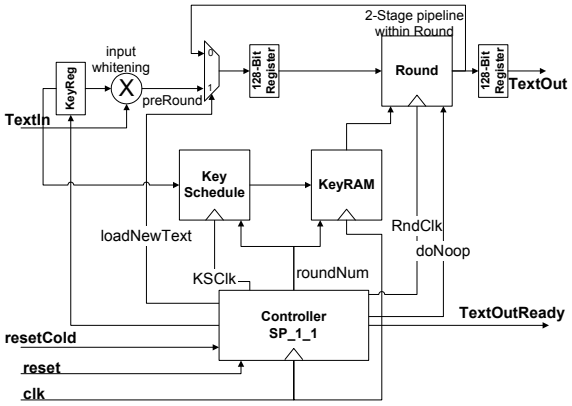

Figure 2: Single sub-pipelined round SP-1-1

The Five-stage loop unrolled implementation needs only a single copy of the key-schedule. This result in less than 4 times area overhead while achieving more than 4 times the throughput of one stage sub-pipelined design.

Table 3: FPGA implementation results of AES

	Area (slices)	Frequency (MHz)	Cycles/ Block	Throughput (Mbps)
SP_1_1	2128	90.55	10.5	1104
SP_5_1	8114	77.69	2.1	4735

Advanced architectures for AES were also implemented on Xilinx XCV1000 FPGA in order to compare them with their previous implementations at Worcester Polytechnic Institute (WPI). Unlike [2] our designs also implement the key scheduling function and use Xtime function to implement MixCol operation. Our implementations showed significant improvement over previous works, with a 50% increase in throughput a 20% decrease in hardware area and these results would improve further if we do not consider the key schedule overheads.

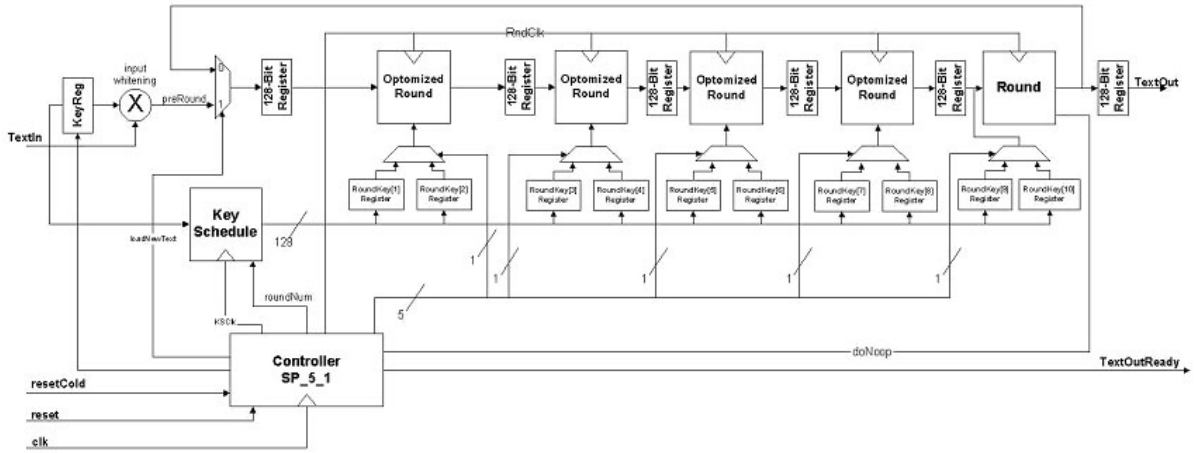

Figure 3: Five-stage sub-pipelined round SP-5-1

3 Stream Ciphers

A stream cipher generates a key stream that is usually combined with plaintext via bitwise exclusive-or (XOR) operation for encryption and decryption. Following sections describe high speed architectures for AES-based, SNOW and Leviathan stream ciphers.

3.1 AES-based Stream Cipher

Certain modes of operation of a block cipher effectively transform it into a key stream generator and as such any block cipher can be used as a stream cipher. One such mode of operation is Integer Counter Mode (ICM) proposed in [4]. The key stream is generated from an ICM key and segment index and XORed with plaintext to generate the corresponding cipher text. Length of the ICM key is dependent on the block-length and the key-length parameters of the underlying block cipher.

The AES block cipher, presented in the prior section, was implemented in ICM mode to create an efficient synchronous key stream generator. Figure 4 shows that the generalized interface of our implementation facilitates easy IP reuse of AES block cipher architectures for AES stream cipher with some trivial modifications. The only additional components needed are the optimized count initializer and a counter with parallel load. Constant addition in parallel-load counter was faster (97MHz) than the block cipher. A multiplexer was used to drive the block cipher input from either the *keyIn* input for the ICM key or the counter.

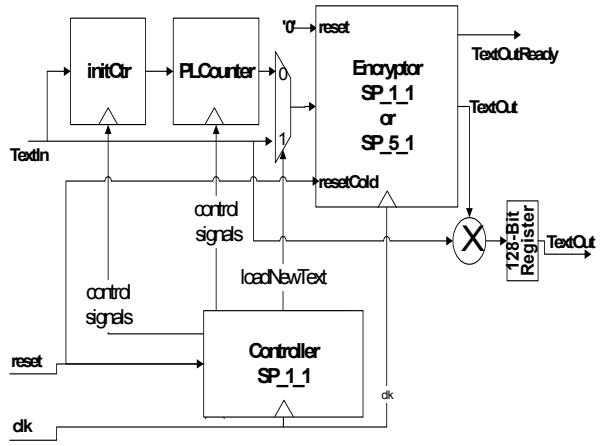

Figure 4: AES Integer Counter Mode stream cipher

ICM counter initialization algorithm required a 96-bit addition which turned out to be the main system bottleneck (see results corresponding to the un-optimized implementation in Table 5). Therefore, ICM counter initialization was optimized by folding the addition onto a single 32-bit carry-select adder operating at 108 MHz. This addition took three cycles, but was performed during key-schedule. Implementation results of these implementations are shown in Table 5.

3.2 SNOW Stream Cipher

A Linear Feedback Shift Register (LFSR) is a mechanism for generating a stream of binary bits [6]. The *register* consists of a series of cells that are initialized using the secret key. At each clocking instant the contents of the cells are shifted right by one position and a non-linear operation (e.g. XOR) is applied to a subset of the cell contents and fed back into the leftmost cell. This is an example of a many-to-1 feedback topology. Another way to implement an LFSR is with a 1-to-many topology where the most significant bit is used in all the taps. Regardless of the topology, LFSRs are fast and easy to implement in both hardware and software and with appropriate *feedback* the generated taps sequences can have a good statistical appearance.

Figure 5 shows SNOW, a LFSR based synchronous stream cipher that supports key sizes of 128 and 256 bits [6]. This architecture does not permit a straightforward tradeoff between area and throughput since it cannot be broken into multiple rounds of computation to facilitate pipelining. Also, since there is a dependency between the generations of consecutive words in the key stream, they cannot be produced in parallel. Critical path consists of two 32-bit additions, two XORs and a left-shift by seven. Therefore in order to increase its performance we investigated a variety of high-speed architectures for adders.

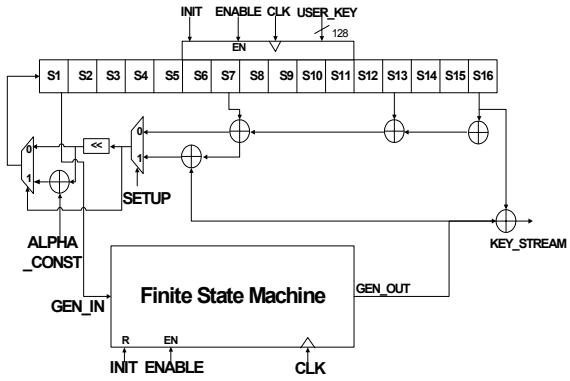

Figure 5: SNOW stream cipher data path

Table 4 shows the results of this analysis based on which it was decided to employ 32-bit carry select adders using 8-bit carry look-ahead adders. Resulting design occupied 752 slices and operated at a frequency of 66.489 MHz. This is approximately three times faster than the fastest software implementation of SNOW stream cipher on Pentium III systems.

Table 4: Comparison of FPGA implementation of 32-bit adders on XC2V2000

	Frequency (MHz)
Ripple-Carry	31.6
Carry Look-ahead	43.9
Carry Look-ahead (8 slices)	76
Carry-Select (4-bit CLA)	72
Carry-Select (8-bit CLA)	89.2

3.3 Leviathan Stream Cipher

Leviathan is a complete binary-tree based stream cipher whose leaves are traversed consecutively from left to right to form a key stream [5]. A straightforward approach for generating 32-bit key values at all the leaf nodes entails O ($h2^h$) computation time and at least O (2^h) storage requirements where h denotes the height of the tree. For example, for a binary tree of height sixteen this translates to 2^{16} 32-bit memory locations for storing keys and O (2^{20}) computation time. In order to reduce this storage requirement key stream should be generated dynamically.

A simple approach for generating the key stream dynamically entails traversing a unique path from the root of the tree to the corresponding leaf node and executing the functions within each node. This scheme yields worst-case computation time which is equal to the height times the execution time of node function and the storage requirement of the order O (1). Period between the generations of consecutive key values is

$T_{key\_period} = \sum T_{A|B(15)} + T_C$

, thereby yielding low throughput.

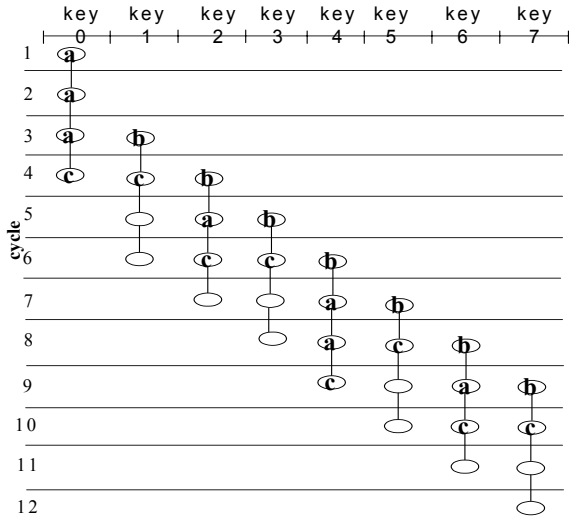

Figure 6: Pipelined traversal of computation tree

We investigated pipelined architectures to process paths leading to consecutive keys. Depth of the pipeline is determined by the height of the tree. This scheme offers fixed period between generations of consecutive keys. Figure 6 shows a tree of height five and its respective pipelined paths. Dataflow in the pipeline traverses the paths from either the intermediate node or the root node to the leaf nodes. Each path from the root to the leaf node is traversed by taking advantage of the shared intermediate nodes. All the node functions along the path traversed from the root to the leaf node are executed in successive stages of the pipeline and their results are stored in the pipeline registers.

Thus, when the data flow of the path traversing to the consecutive leaf node is pipelined these values need not be re-computed. This architecture can also efficiently implement power-aware pipeline in which clock gating is performed to disable specific data paths during empty pipeline slots.

3.4 FPGA Implementation of Stream Ciphers

Implementation results of the various stream ciphers on the Virtex-II series FPGAs are presented in Table 5. Optimized AES stream cipher operates at the same speed as the AES block cipher (Table 3), though it consumes 5% more area. 32-bit data path SNOW throughput approaches 3 Gbps. Expansion of this design to a 128-bit data path promises throughputs in the 12Gbps range.

Table 5: FPGA implementation results of stream ciphers

		Area (slices)	Frequency (MHz)	Throughput (Mbps)
AES ICM SP-1-1	Un-optimized	2282	74.251	905.155
	Optimized	2512	94.02	1146.149
AES ICM SP-5-1	Un-optimized	8326	40.025	2439.62
	Optimized	8631	76.159	4642.072
SNOW		752	66.489	2812.52

4 Conclusion

Various high-speed architectures of encryption primitives were discussed to satisfy the rapidly increasing demands of broadband data networks. Optimized FPGA implementations of AES block and stream ciphers and SNOW stream cipher with high throughput rates of 4.7 Gbps, 4.6 Gbps and 2.12 Gbps respectively were presented. Schemes exploiting parallelism cannot be directly extended to ciphers operating in the feedback modes such as CBC, CFB, and OFB. On the other hand, decryption in these feedback modes can utilize these high-speed architectures since they allow data path parallelism.

An analysis of the work under progress on the pipelined implementations of Leviathan stream cipher with optimized storage and computation requirements was also presented.

5 References

[1] J. Daemen, V. Rijmen, "AES proposal: Rijndael," *First Advanced Encryption Standard (AES) Conference*, CA, USA, 1998.

[2] A. J. Elbirt, W. Yip, B. Chetwynd, C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transactions on VLSI Systems*, Vol. 9, No. 4, pp. 545–557, August 2001.

[3] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Concurrent error detection of fault-based side-channel cryptanalysis of 128-bit symmetric block ciphers," *Proceedings, IEEE Design Automation Conference (DAC)*, NV, USA, June 2001.

[4] H. Lipmaa, P. Rogaway, and D. Wagner, "Comments to NIST concerning AES modes of operations: CTR-mode encryption," Symmetric Key Block Cipher Modes of Operation Workshop, MD, USA, October 2000.

[5] D. A. McGrew and S. R. Fluhrer "The stream cipher Leviathan," *New European Schemes for Signatures, Integrity and Encryption (NESSIE)*, October 2000.

[6] P. Ekdahl and T. Johansson, "SNOW: A new stream cipher," *Department of Information Technology, Lund University*, November, 2001.

[7] Federal Information Processing Standards (FIPS), "Announcing the Advanced Encryption Standard (AES)," *National Institute of Standards and Technology (NIST)*, November 2001.

[8] D. W. Davis and W. L. Price, "Security for Computer Networks," *Wiley*, 1989.

[9] HiFn Corporation. http://hifn.com

[10] S/390 and OS/390 Cryptography. http://www.s390.ibm.com/security/cryptography.html

[11] Shiva Corporation. http:/shiva/com

[12] X. Lai, "On the Design and Security of Block Ciphers," *Hartung-Gorre Veerlag*, 1992

[13] L. Wu, C. Weaver, and T. Austin, "CryptoManiac: A fast flexible architecture for secure communication," *International Symposium on Computer Architecture (ISCA)*, Sweden, June 2001.